FIG. 1A
PRIOR ART

```
_asm int uci(int_reg_src, unsigned char_imm);        .....(F91)
          ⋮

a = uci(b, 255);                                     .....(T91)

          ⋮

a = (b+10) & 255;                                    .....(T92)
```

FIG. 1B
PRIOR ART

```
uci $0, $1, 255                                      .....(M91)

          ⋮

add $3, $1, 10                                      }
add $0, $3, 255                                     } .....(M92)
```

FIG. 2

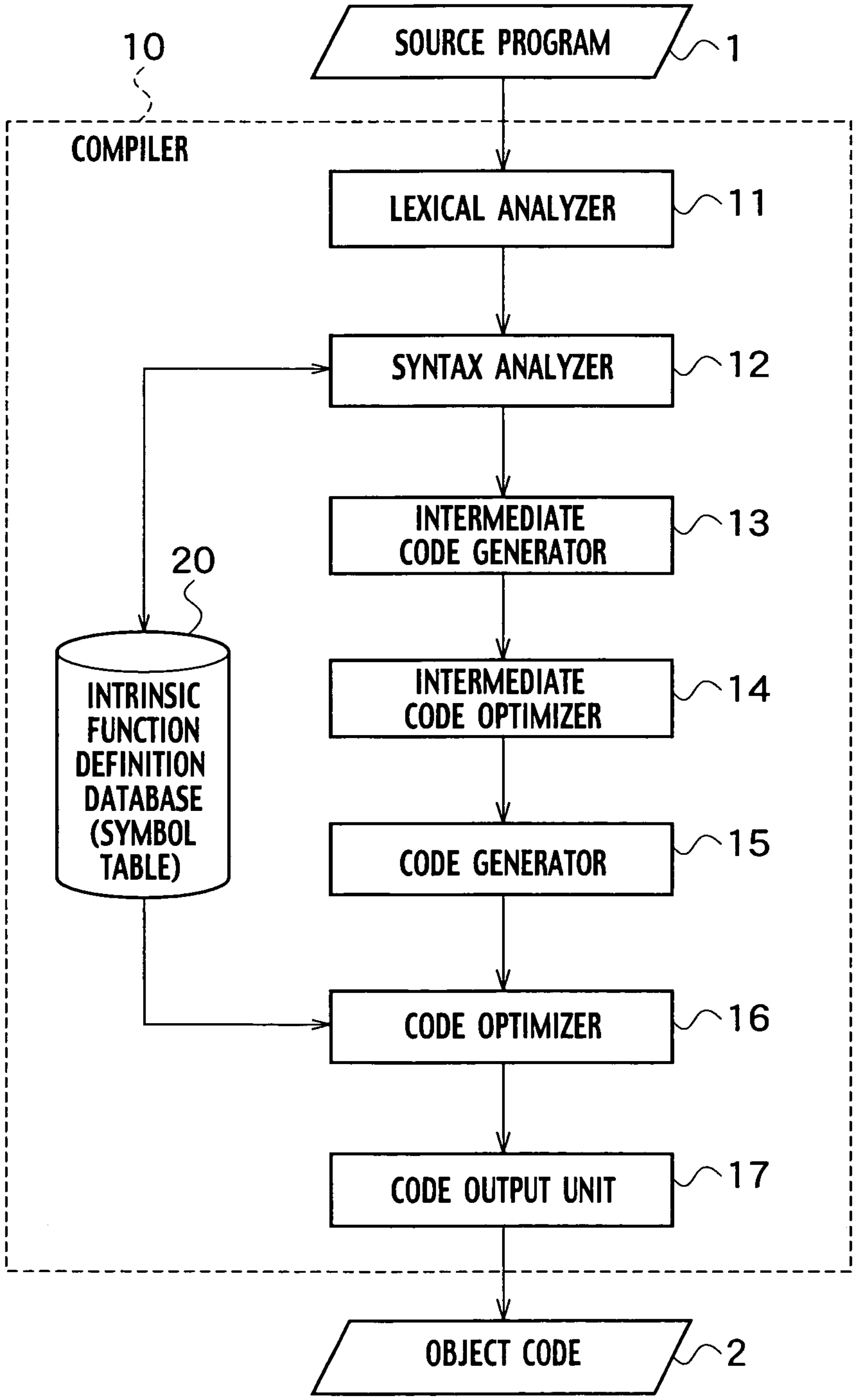

SOURCE PROGRAM
1
10
COMPILER
LEXICAL ANALYZER
11
SYNTAX ANALYZER
12
INTERMEDIATE CODE GENERATOR
13
20
INTRINSIC FUNCTION DEFINITION DATABASE (SYMBOL TABLE)
INTERMEDIATE CODE OPTIMIZER
14
CODE GENERATOR
15
CODE OPTIMIZER
16
CODE OUTPUT UNIT
17
OBJECT CODE
2

FIG. 3

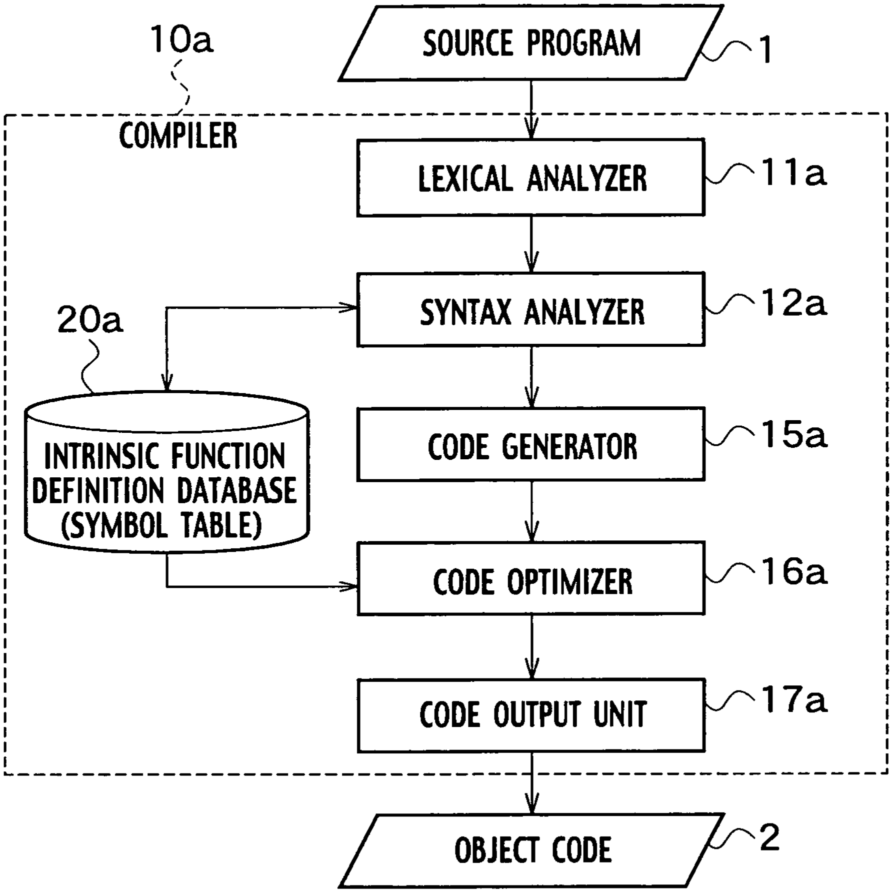

SOURCE PROGRAM
1
10a
COMPILER
LEXICAL ANALYZER
11a
SYNTAX ANALYZER
12a
20a
INTRINSIC FUNCTION DEFINITION DATABASE (SYMBOL TABLE)
CODE GENERATOR
15a
CODE OPTIMIZER
16a
CODE OUTPUT UNIT
17a
OBJECT CODE
2

FIG. 4

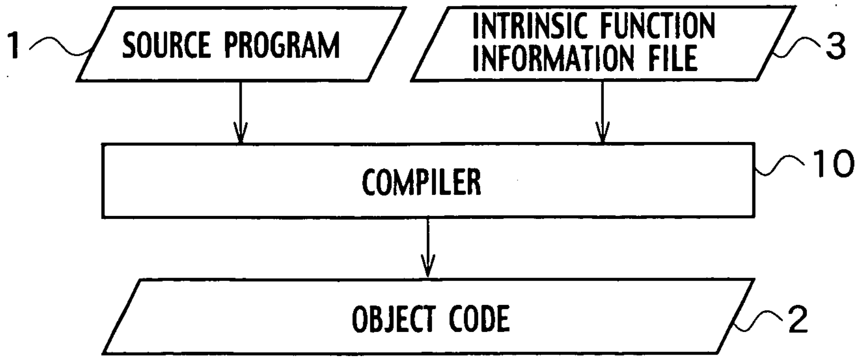
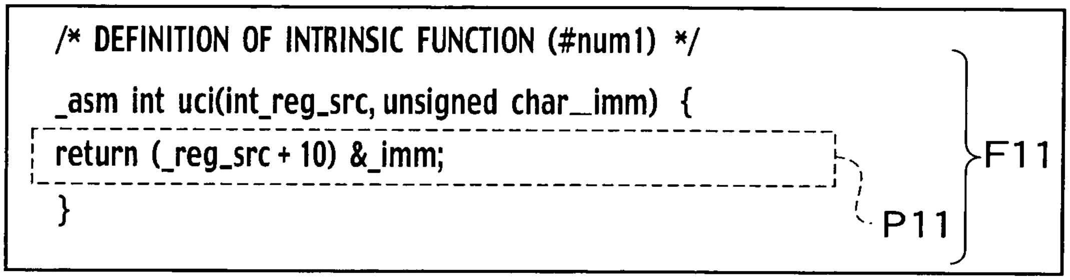
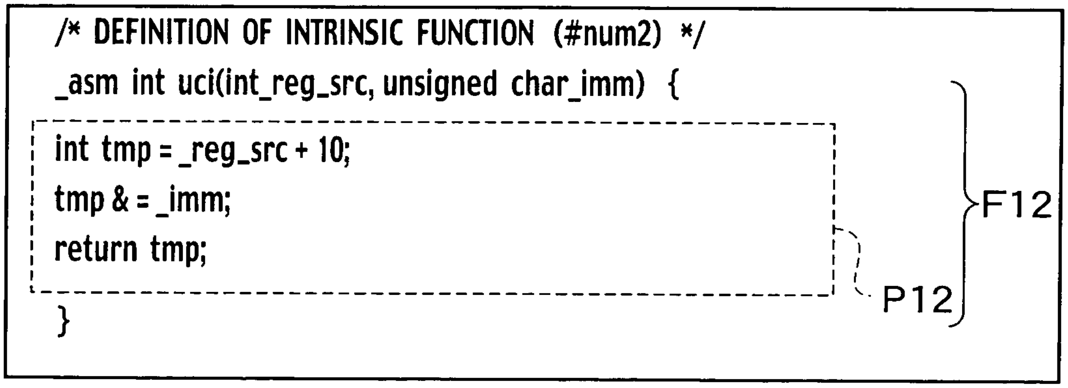

1
SOURCE PROGRAM
INTRINSIC FUNCTION INFORMATION FILE
3
COMPILER
10
OBJECT CODE
2

FIG. 5A



```
/* DEFINITION OF INTRINSIC FUNCTION (#num1) */
_asm int uci(int_reg_src, unsigned char_imm) {
return (_reg_src + 10) &_imm;
}
```

FIG. 5B



```
/* DEFINITION OF INTRINSIC FUNCTION (#num2) */
_asm int uci(int_reg_src, unsigned char_imm) {
int tmp = _reg_src + 10;
tmp & = _imm;
return tmp;
}
```

FIG. 6A

```
/* EXPLICIT CALL OF INTRINSIC FUNCTION */
int a, b;
a = uci( b, 255);                         .....(T11)
a = uci(a, 127);                          .....(T12)
```

FIG. 6B

```
uci $0, $1, 255                           .....(M11)
uci $0, $0, 127                           .....(M12)
```

FIG. 7A

```
int a, b;
a = (b+10) &255;                      .....(T21)
a = (a+10) &127;                      .....(T22)
```

FIG. 7B

```
add $3, $1, 10                        }
add $0, $3, 255                       } .....(M21)
add $4, $0, 10                        }
add $0, $4, 127                       } .....(M22)
```

FIG. 7C

```
uci $0, $1, 255                       .....(M23)
uci $0, $0, 127                       .....(M24)
```

FIG. 8

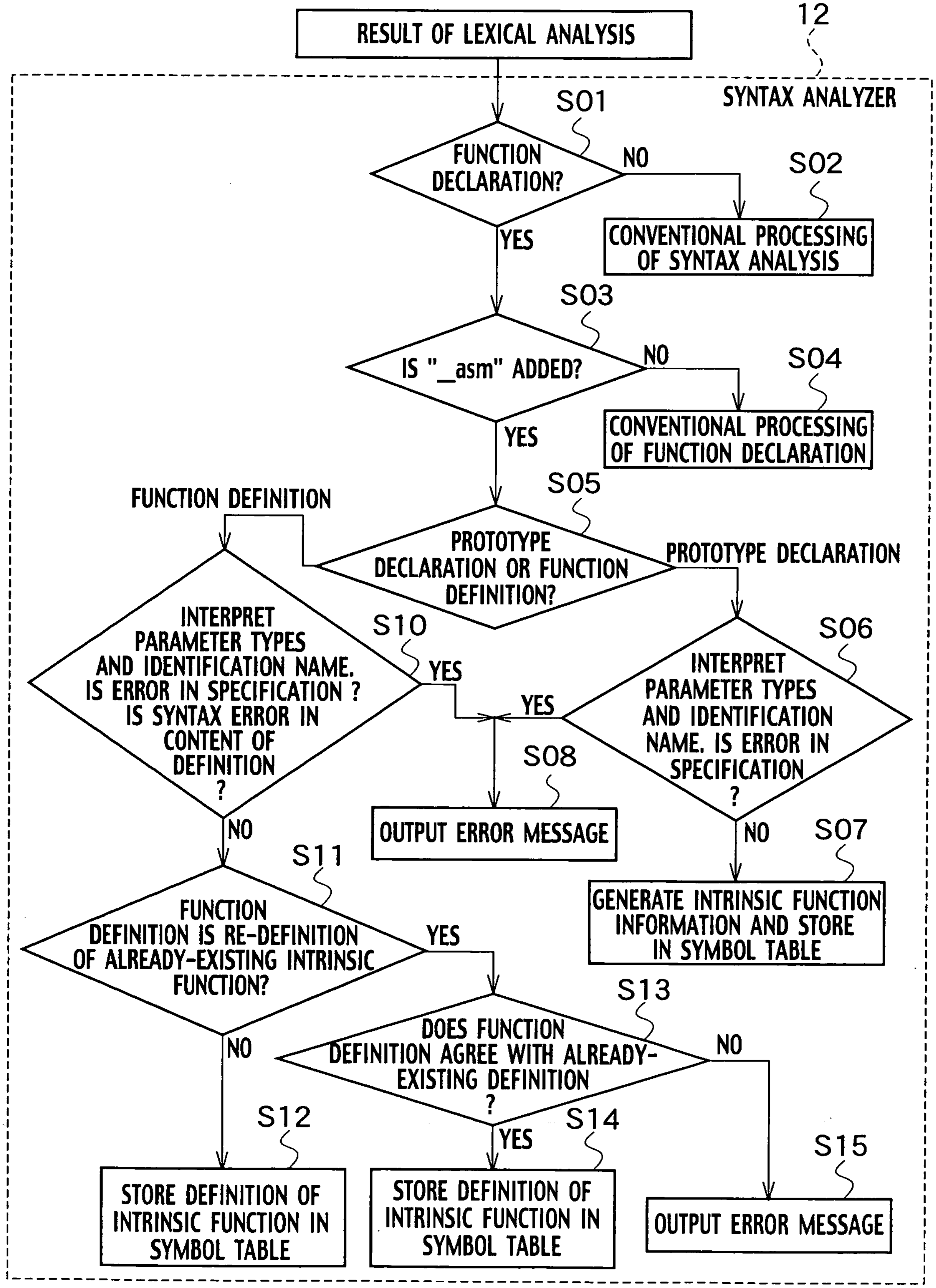
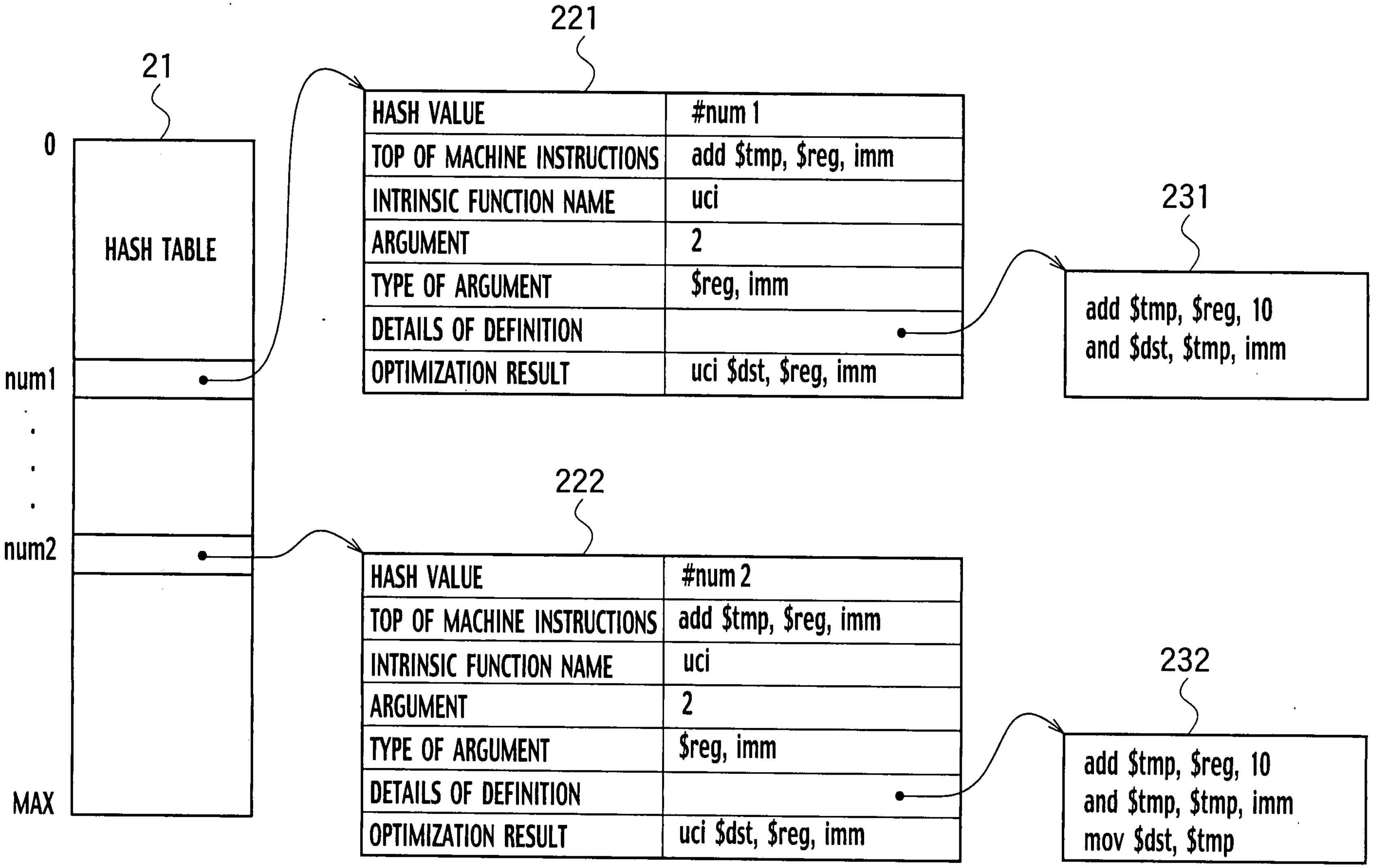

RESULT OF LEXICAL ANALYSIS
12
SYNTAX ANALYZER
S01
FUNCTION DECLARATION?
NO
S02
CONVENTIONAL PROCESSING OF SYNTAX ANALYSIS
YES
S03
IS "_asm" ADDED?
NO
S04
CONVENTIONAL PROCESSING OF FUNCTION DECLARATION
YES
S05
FUNCTION DEFINITION
PROTOTYPE DECLARATION OR FUNCTION DEFINITION?
PROTOTYPE DECLARATION
INTERPRET PARAMETER TYPES AND IDENTIFICATION NAME. IS ERROR IN SPECIFICATION ? IS SYNTAX ERROR IN CONTENT OF DEFINITION ?
S10
YES
YES
S06
INTERPRET PARAMETER TYPES AND IDENTIFICATION NAME. IS ERROR IN SPECIFICATION ?
S08
OUTPUT ERROR MESSAGE
NO
NO
S07
GENERATE INTRINSIC FUNCTION INFORMATION AND STORE IN SYMBOL TABLE
S11
FUNCTION DEFINITION IS RE-DEFINITION OF ALREADY-EXISTING INTRINSIC FUNCTION?
YES
S13
DOES FUNCTION DEFINITION AGREE WITH ALREADY-EXISTING DEFINITION ?
NO
NO
YES
S12
S14
S15
STORE DEFINITION OF INTRINSIC FUNCTION IN SYMBOL TABLE
STORE DEFINITION OF INTRINSIC FUNCTION IN SYMBOL TABLE
OUTPUT ERROR MESSAGE

FIG. 9



HASH VALUE	#num 1
TOP OF MACHINE INSTRUCTIONS	add $tmp, $reg, imm
INTRINSIC FUNCTION NAME	uci
ARGUMENT	2
TYPE OF ARGUMENT	$reg, imm
DETAILS OF DEFINITION	●
OPTIMIZATION RESULT	uci $dst, $reg, imm

```
add $tmp, $reg, 10
and $dst, $tmp, imm
```

HASH VALUE	#num 2
TOP OF MACHINE INSTRUCTIONS	add $tmp, $reg, imm
INTRINSIC FUNCTION NAME	uci
ARGUMENT	2
TYPE OF ARGUMENT	$reg, imm
DETAILS OF DEFINITION	●
OPTIMIZATION RESULT	uci $dst, $reg, imm

```
add $tmp, $reg, 10
and $tmp, $tmp, imm
mov $dst, $tmp
```

FIG. 10

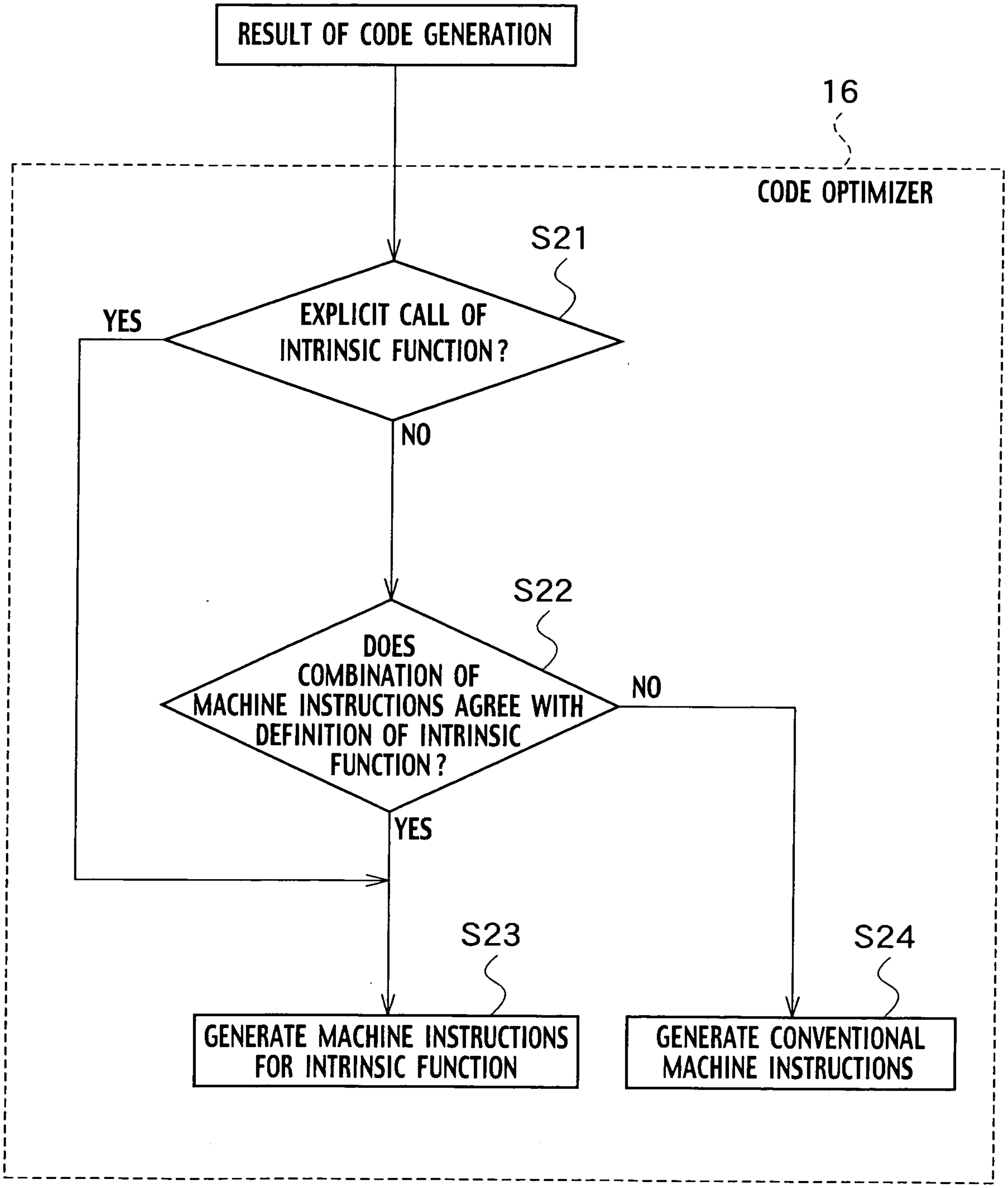
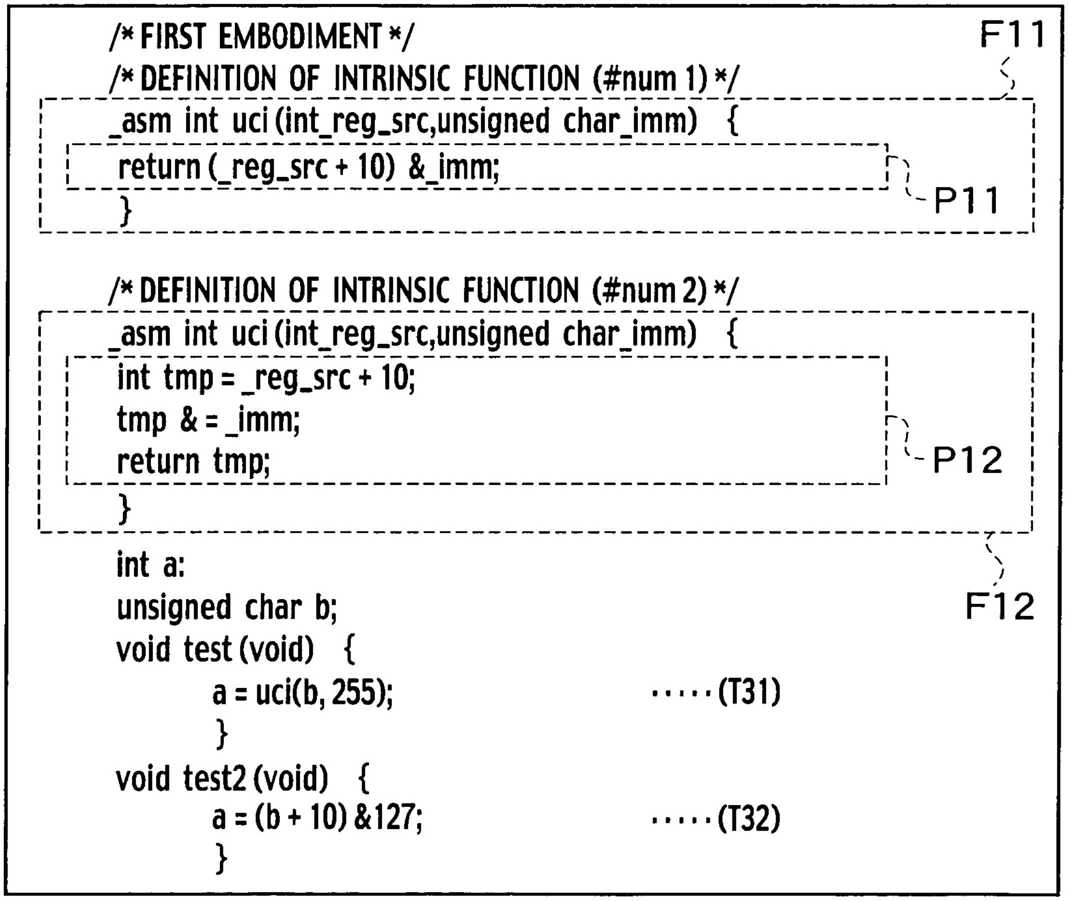

RESULT OF CODE GENERATION
16
CODE OPTIMIZER
S21
EXPLICIT CALL OF INTRINSIC FUNCTION?
YES
NO
S22
DOES COMBINATION OF MACHINE INSTRUCTIONS AGREE WITH DEFINITION OF INTRINSIC FUNCTION?
NO
YES
S23
GENERATE MACHINE INSTRUCTIONS FOR INTRINSIC FUNCTION
S24
GENERATE CONVENTIONAL MACHINE INSTRUCTIONS

FIG. 11

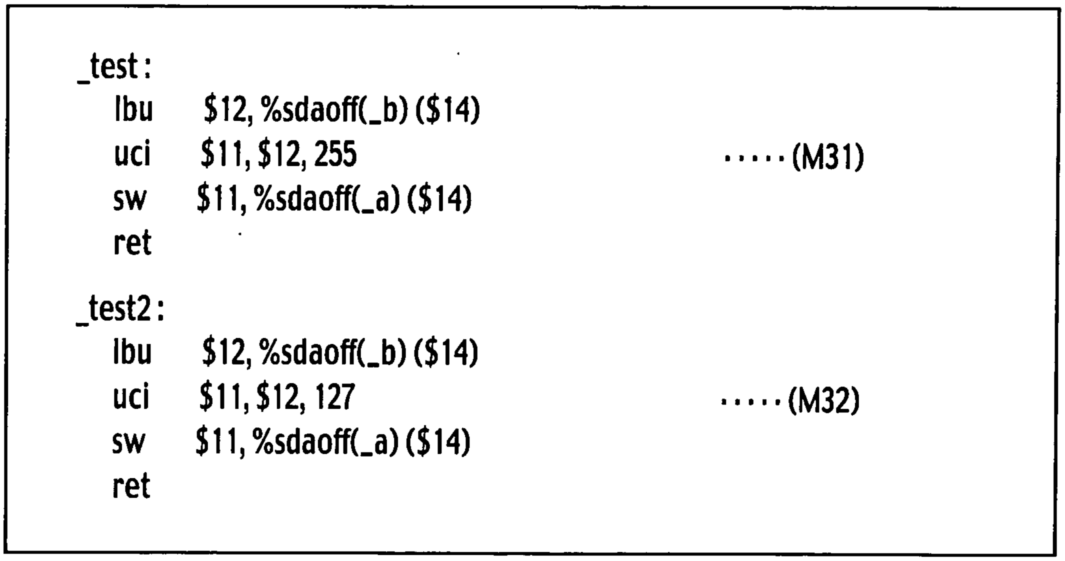


```
    /* FIRST EMBODIMENT */
    /* DEFINITION OF INTRINSIC FUNCTION (#num 1) */
    _asm int uci (int_reg_src,unsigned char_imm)  {
     return (_reg_src + 10) &_imm;
    }

    /* DEFINITION OF INTRINSIC FUNCTION (#num 2) */
    _asm int uci (int_reg_src,unsigned char_imm)  {
     int tmp = _reg_src + 10;
     tmp & = _imm;
     return tmp;
    }
     int a:
     unsigned char b;
     void test (void)  {
             a = uci(b, 255);                 .....(T31)
             }
     void test2 (void)  {
             a = (b + 10) &127;               .....(T32)
             }
```

F11
P11
P12
F12

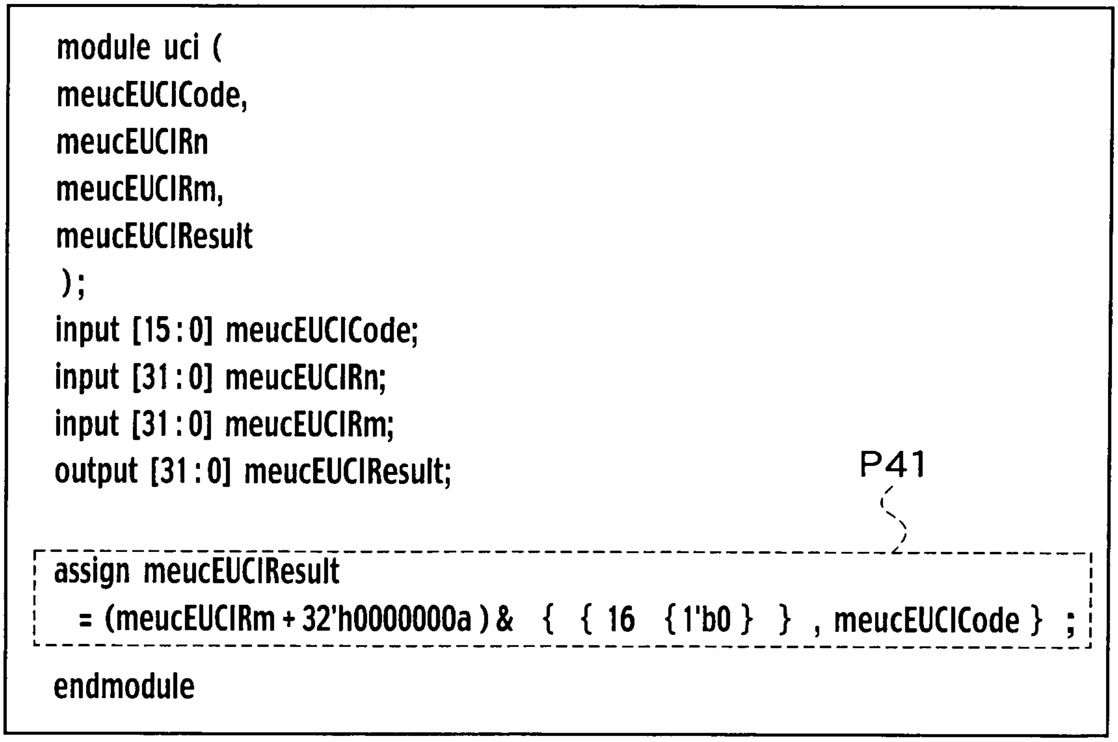
FIG. 12



```
_test:
   lbu    $12, %sdaoff(_b) ($14)
   uci    $11, $12, 255                        ·····(M31)
   sw     $11, %sdaoff(_a) ($14)
   ret

_test2:
   lbu    $12, %sdaoff(_b) ($14)
   uci    $11, $12, 127                        ·····(M32)
   sw     $11, %sdaoff(_a) ($14)
   ret
```

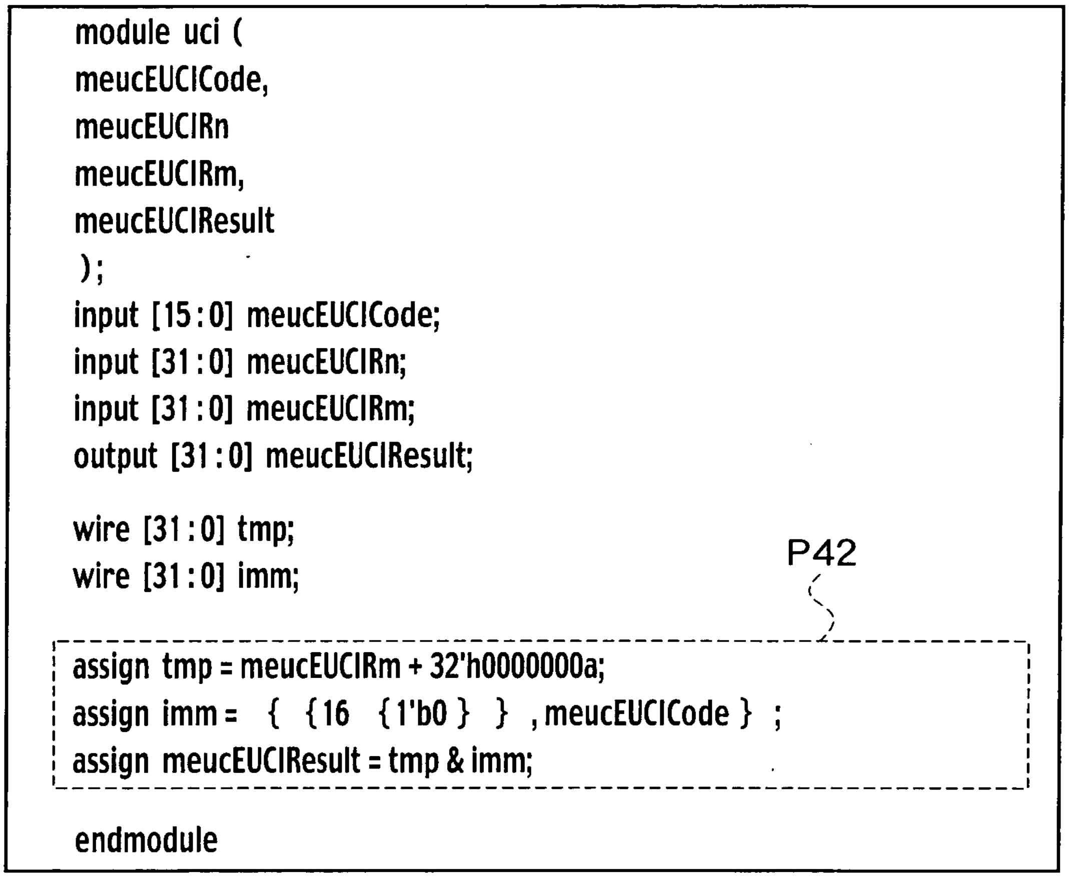
FIG. 13



```
module uci (
meucEUCICode,
meucEUCIRn
meucEUCIRm,
meucEUCIResult
);
input [15:0] meucEUCICode;
input [31:0] meucEUCIRn;
input [31:0] meucEUCIRm;
output [31:0] meucEUCIResult;

assign meucEUCIResult
  = (meucEUCIRm + 32'h0000000a) & { { 16 {1'b0} } , meucEUCICode } ;
endmodule
```

FIG. 14



```
module uci (
meucEUCICode,
meucEUCIRn
meucEUCIRm,
meucEUCIResult
);
input [15:0] meucEUCICode;
input [31:0] meucEUCIRn;
input [31:0] meucEUCIRm;
output [31:0] meucEUCIResult;

wire [31:0] tmp;
wire [31:0] imm;

assign tmp = meucEUCIRm + 32'h0000000a;
assign imm =  {  {16  {1'b0 }  }  , meucEUCICode }  ;
assign meucEUCIResult = tmp & imm;

endmodule
```

FIG. 15A

```
/* SECOND EMBODIMENT */
#pragma input HDL add10_and_1.V                ·····(H41)
#pragma input HDL add10_and_2.V                ·····(H42)
int a:
unsigned char b;
void test(void)  {
    a = uci( b, 255):                          ·····(T41)
     }
void test2(void)  {
    a = (b + 10) & 127:                        ·····(T42)
    }
```

FIG. 15B

```
_test:
   lbu     $12, %sdaoff(_b) ($14)
   uci     $11, $12, 255                       ·····(M41)
   sw     $11, %sdaoff(_a) ($14)
   ret
_test2:
   lbu     $12, %sdaoff(_b) ($14)
   uci     $11, $12, 127                       ·····(M42)
   sw     $11, %sdaoff(_a) ($14)
   ret
```

FIG. 16

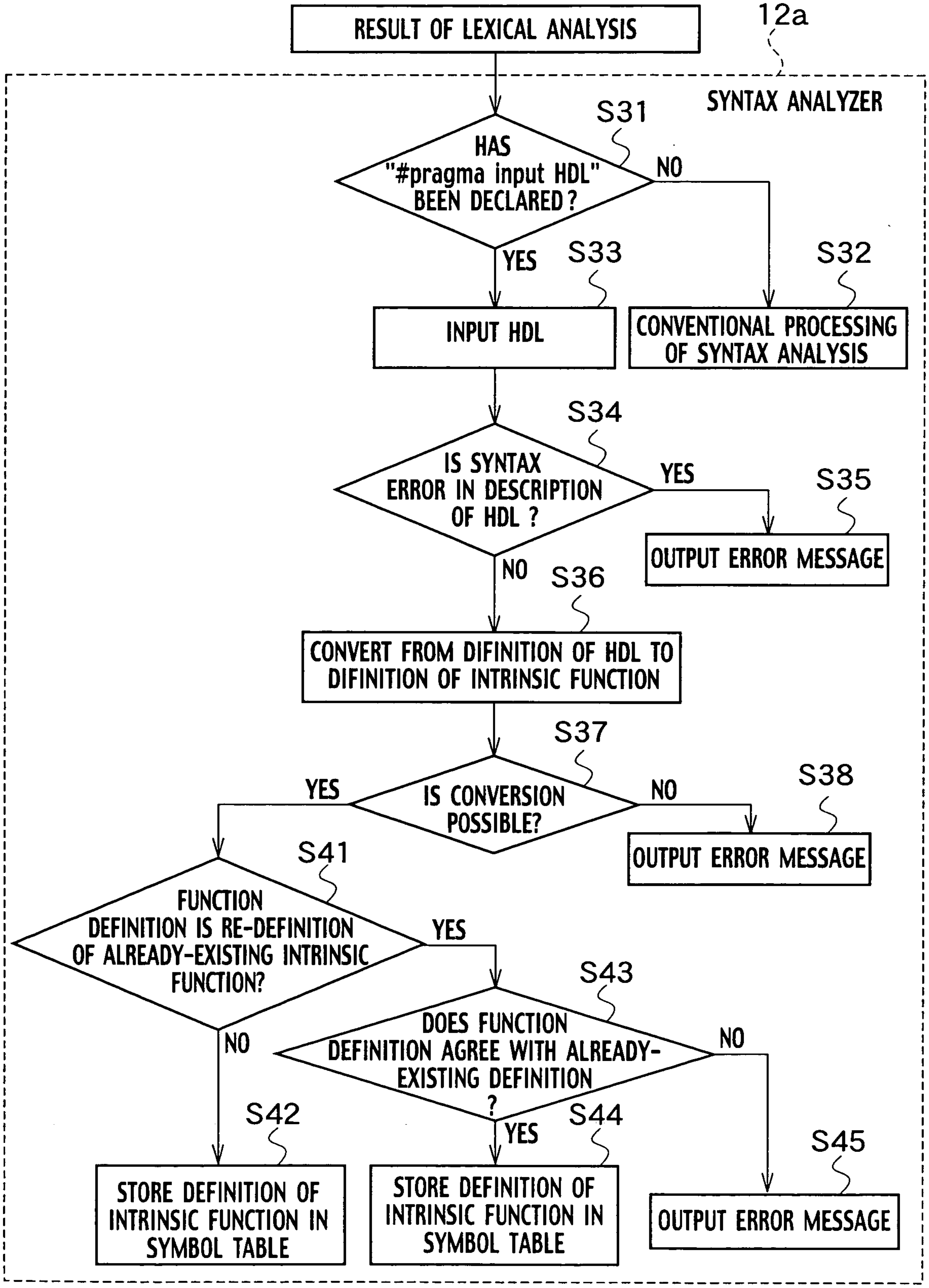
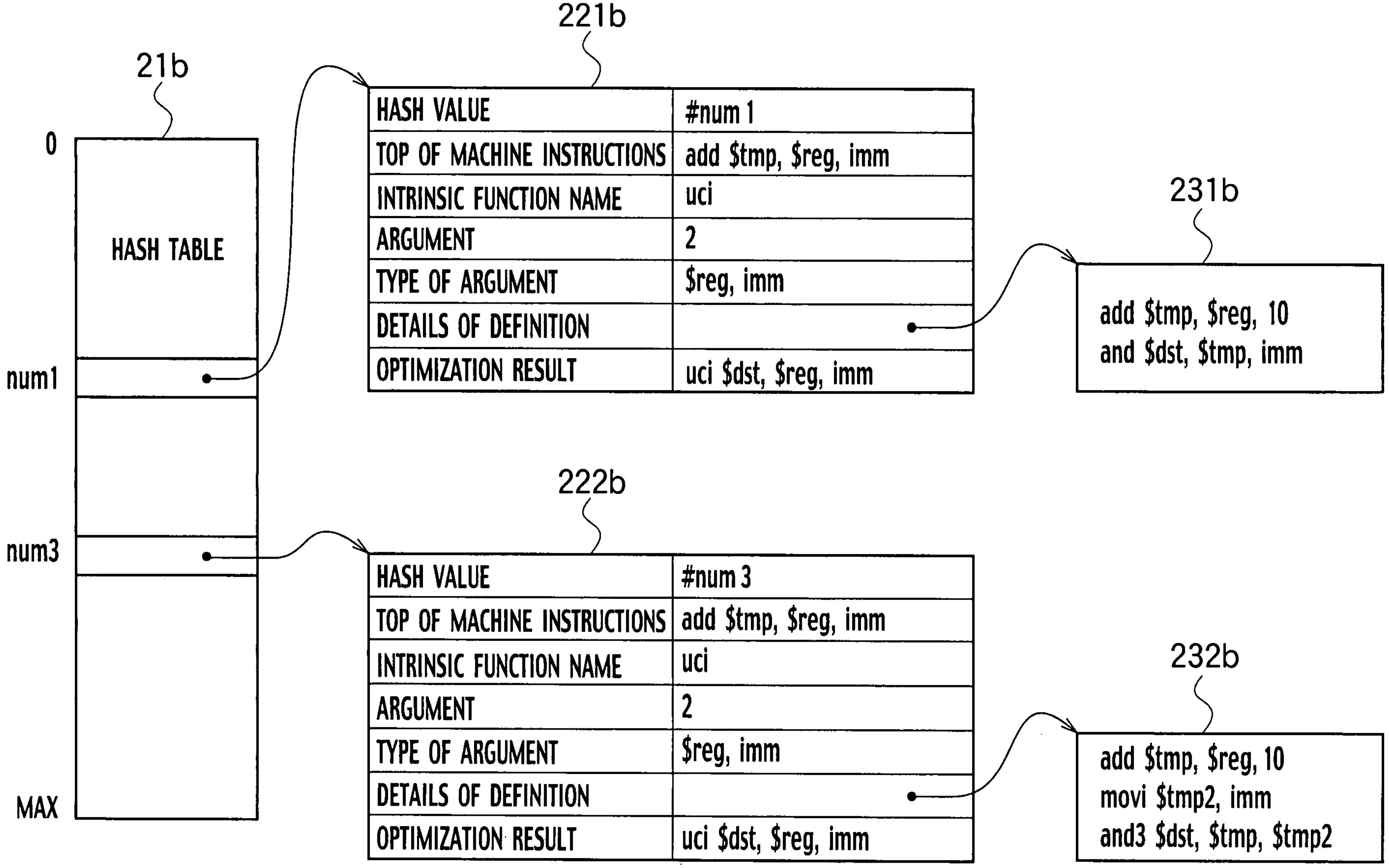

RESULT OF LEXICAL ANALYSIS
12a
SYNTAX ANALYZER
S31
HAS "#pragma input HDL" BEEN DECLARED?
NO
YES
S33
INPUT HDL
S32
CONVENTIONAL PROCESSING OF SYNTAX ANALYSIS
S34
IS SYNTAX ERROR IN DESCRIPTION OF HDL ?
YES
S35
OUTPUT ERROR MESSAGE
NO
S36
CONVERT FROM DIFINITION OF HDL TO DIFINITION OF INTRINSIC FUNCTION
S37
YES
IS CONVERSION POSSIBLE?
NO
S38
OUTPUT ERROR MESSAGE
S41
FUNCTION DEFINITION IS RE-DEFINITION OF ALREADY-EXISTING INTRINSIC FUNCTION?
YES
S43
NO
DOES FUNCTION DEFINITION AGREE WITH ALREADY-EXISTING DEFINITION ?
NO
YES
S42
S44
S45
STORE DEFINITION OF INTRINSIC FUNCTION IN SYMBOL TABLE
STORE DEFINITION OF INTRINSIC FUNCTION IN SYMBOL TABLE
OUTPUT ERROR MESSAGE

FIG. 17



221b

HASH VALUE	#num 1
TOP OF MACHINE INSTRUCTIONS	add $tmp, $reg, imm
INTRINSIC FUNCTION NAME	uci
ARGUMENT	2
TYPE OF ARGUMENT	$reg, imm
DETAILS OF DEFINITION	●
OPTIMIZATION RESULT	uci $dst, $reg, imm

231b

```
add $tmp, $reg, 10
and $dst, $tmp, imm
```

222b

HASH VALUE	#num 3
TOP OF MACHINE INSTRUCTIONS	add $tmp, $reg, imm
INTRINSIC FUNCTION NAME	uci
ARGUMENT	2
TYPE OF ARGUMENT	$reg, imm
DETAILS OF DEFINITION	●
OPTIMIZATION RESULT	uci $dst, $reg, imm

232b

```
add $tmp, $reg, 10
movi $tmp2, imm
and3 $dst, $tmp, $tmp2
```

FIG. 18

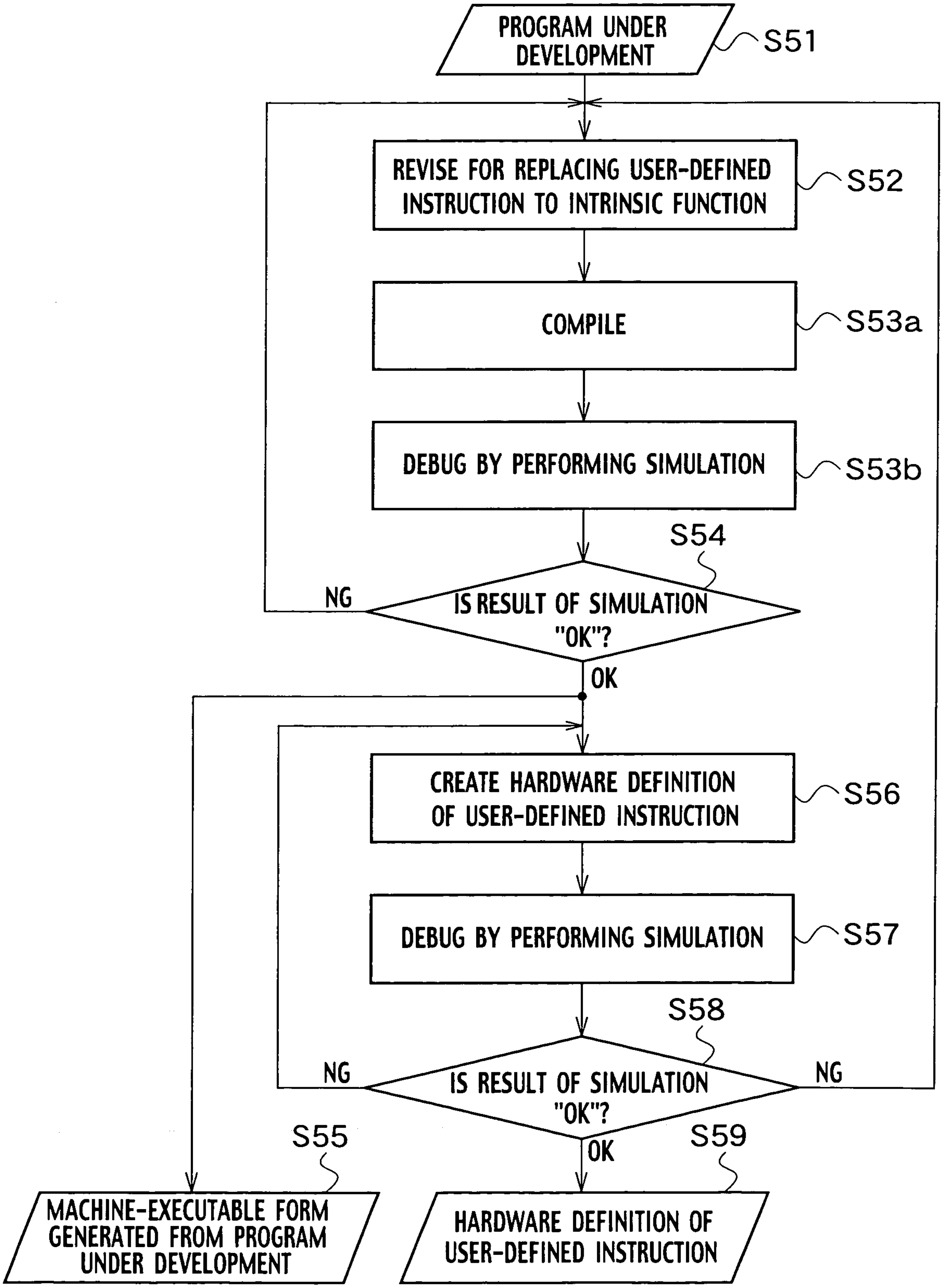
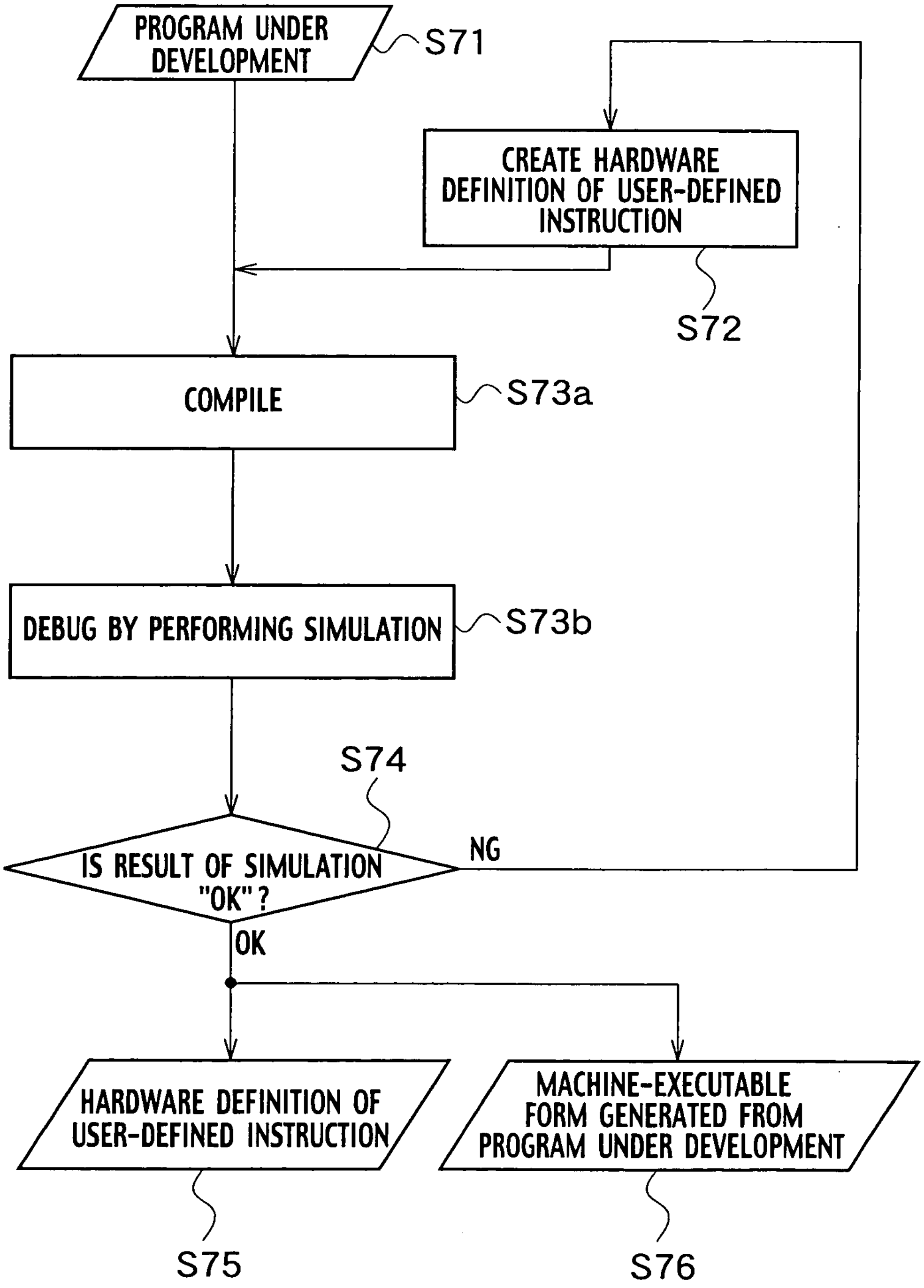

PROGRAM UNDER DEVELOPMENT
S51
REVISE FOR REPLACING USER-DEFINED INSTRUCTION TO INTRINSIC FUNCTION
S52
COMPILE
S53a
DEBUG BY PERFORMING SIMULATION
S53b
S54
NG
IS RESULT OF SIMULATION "OK"?
OK
CREATE HARDWARE DEFINITION OF USER-DEFINED INSTRUCTION
S56
DEBUG BY PERFORMING SIMULATION
S57
S58
NG
IS RESULT OF SIMULATION "OK"?
NG
S55
OK
S59
MACHINE-EXECUTABLE FORM GENERATED FROM PROGRAM UNDER DEVELOPMENT
HARDWARE DEFINITION OF USER-DEFINED INSTRUCTION

FIG. 19


PROGRAM UNDER DEVELOPMENT
S71
CREATE HARDWARE DEFINITION OF USER-DEFINED INSTRUCTION
S72
COMPILE
S73a
DEBUG BY PERFORMING SIMULATION
S73b
S74
IS RESULT OF SIMULATION "OK"?
NG
OK
HARDWARE DEFINITION OF USER-DEFINED INSTRUCTION
MACHINE-EXECUTABLE FORM GENERATED FROM PROGRAM UNDER DEVELOPMENT
S75
S76